# Entwicklung eines Intel HD Audio Sounkartentreibers in Rust

## Präsentation zur Bachelorarbeit

**Sebastian Heidelberg, 03.06.2024**

# Übersicht

- Zielsetzung
- Grundlagen: Rust, D³OS, PCI, PCM, Intel HD Audio
- Implementierung: Struktur des Treibers, Integration in das D³OS, Detailbetrachtung der Module des Treibers
- Evaluation: Herausforderungen bei der Treiberentwicklung, Funktionalität des Treibers, Demo
- Fazit und Ausblick: Zusammenfassung, Mögliche Erweiterungen

# Motivation

## Einleitung

- Treiber sind geräteabhängig und betriebssystemspezifisch
- Das D³OS besitzt noch keinen Intel HD Audio Soundkartentreiber

# Zielsetzung

## Einleitung

- Bereitstellung einer API zur Bedienung einer Intel HD Audio Soundkarte durch das D³OS
- Wiedergabe eines im Systemspeicher definierten Audiosignals über einen analogen Ausgang der Soundkarte mittels dieser API

# Rust

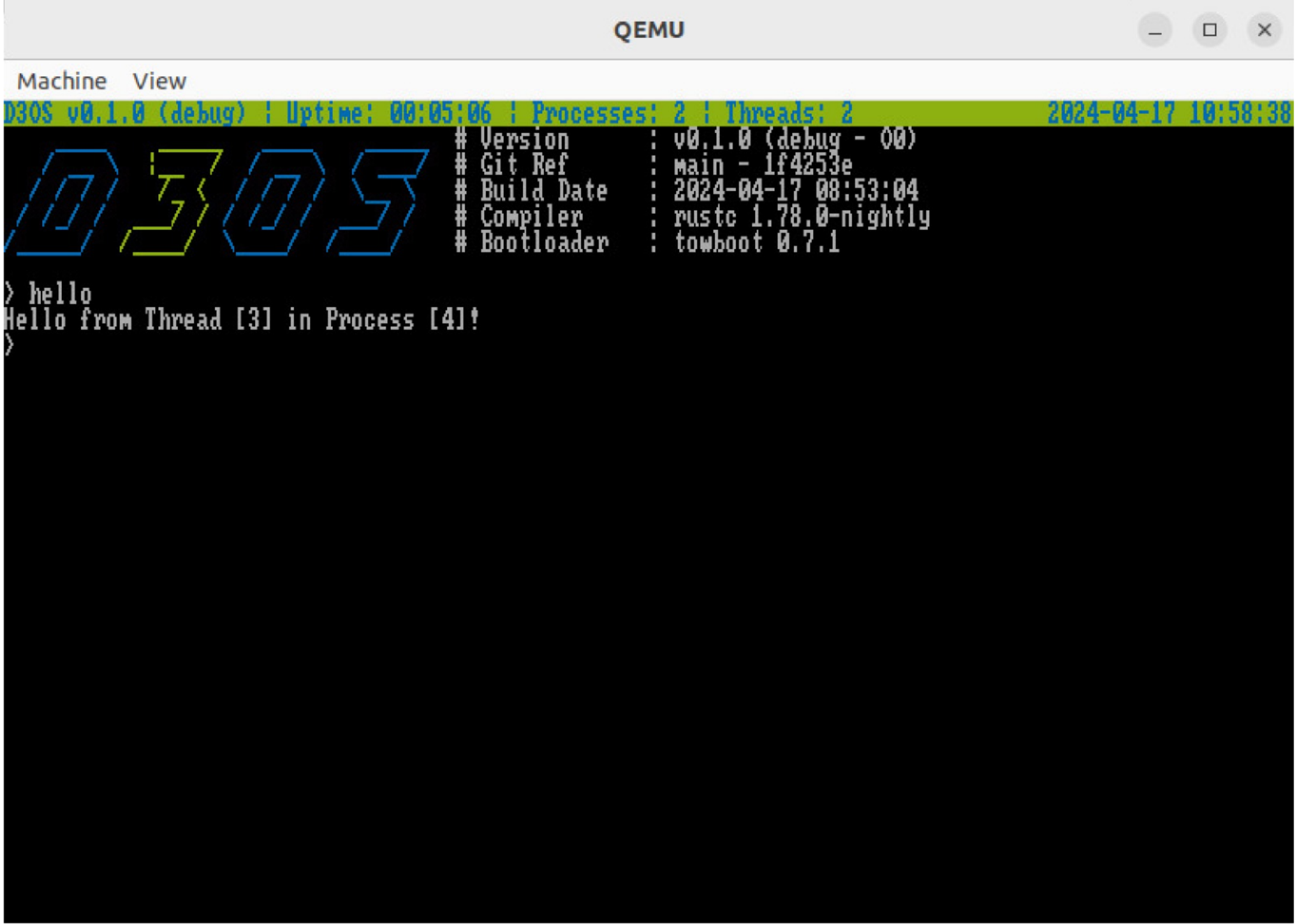
## Grundlagen

- Rust 1.0 released am 15.05.2015 → relativ junge Sprache
- System-Programmiersprache → manuelle Speicherverwaltung (für Dereferenzierung von Zeigern unsicheres Rust notwendig, wichtig für DMA)
- Performante High-Level-Abstractions (Traits, Closures, …)
- Cargo: Build-System und Package-Manager (Packages, Crates, Module)

# D³OS

## Grundlagen

- Distributed Data-Driven OS wird seit 2023 in der Abteilung Betriebssysteme entwickelt
- Interrupt-Handling, Heap, PS/2-Tastaturtreiber, Programm „hello“
- Virtuelle Speicherverwaltung: Soundkarte greift auf physischen Adressraum, Treiber auf Kernel-Adressraum zu


QEMU
Machine View
D3OS v0.1.0 (debug) | Uptime: 00:05:06 | Processes: 2 | Threads: 2 2024-04-17 10:58:38
# Version : v0.1.0 (debug - 00)
# Git Ref : main - 1f4253e
# Build Date : 2024-04-17 08:53:04
# Compiler : rustc 1.78.0-nightly
# Bootloader : towboot 0.7.1
> hello
Hello from Thread [3] in Process [4]!
>

# PCI

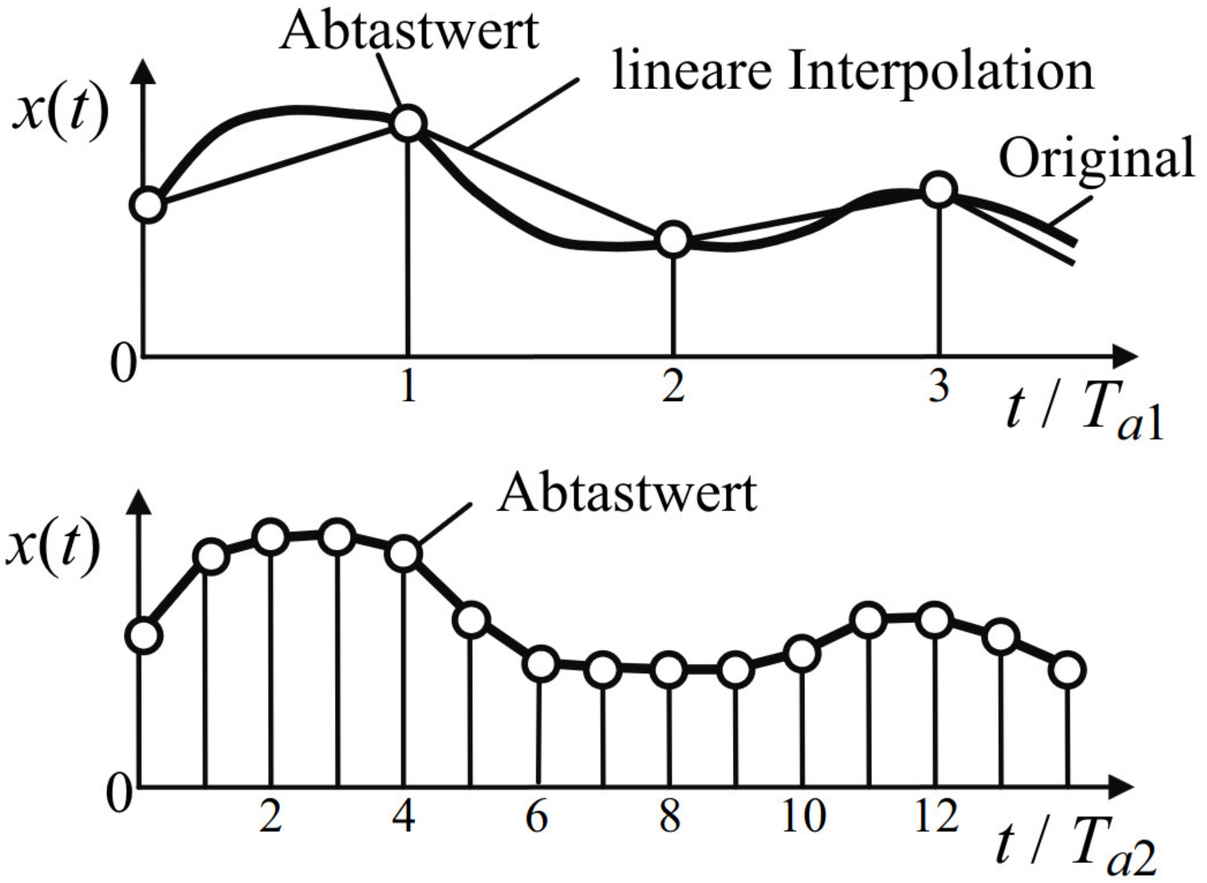
## Grundlagen

- Peripheral Component Interconnect: Bus-Standard zur Verbindung von Geräten
- Jedes Gerät besitzt PCI Configuration Space mit Status- und Kontrollregistern
- Intel HD Audio Soundkarten sind i. d. R. über PCI an einen Rechner angeschlossen

# PCM

## Grundlagen

- Pulse-Code-Modulation: Standardverfahren zur Digitalisierung von Audiosignalen
- Umwandlung eines zeitkontinuierlichen Signals (analog) in ein zeitdiskretes (digital)
- DAC/ADC, Sample, Samplerate (z.B. 48kHz), Bittiefe (z.B. 16 Bit)




Nachrichtentechnik, eine Einführung für alle Studiengänge, Vieweg+Teubner, 2006, Bild 3-10, S. 68

# Intel HD Audio

## Grundlagen

- Intel HD Audio Spezifikation Revision 1.0a vom 17.06.2010
- Definiert Architektur und Programmiermodell einer Intel HD Audio Soundkarte
  → richtet sich sowohl an Hersteller von Hardware als auch an Treiberentwickler

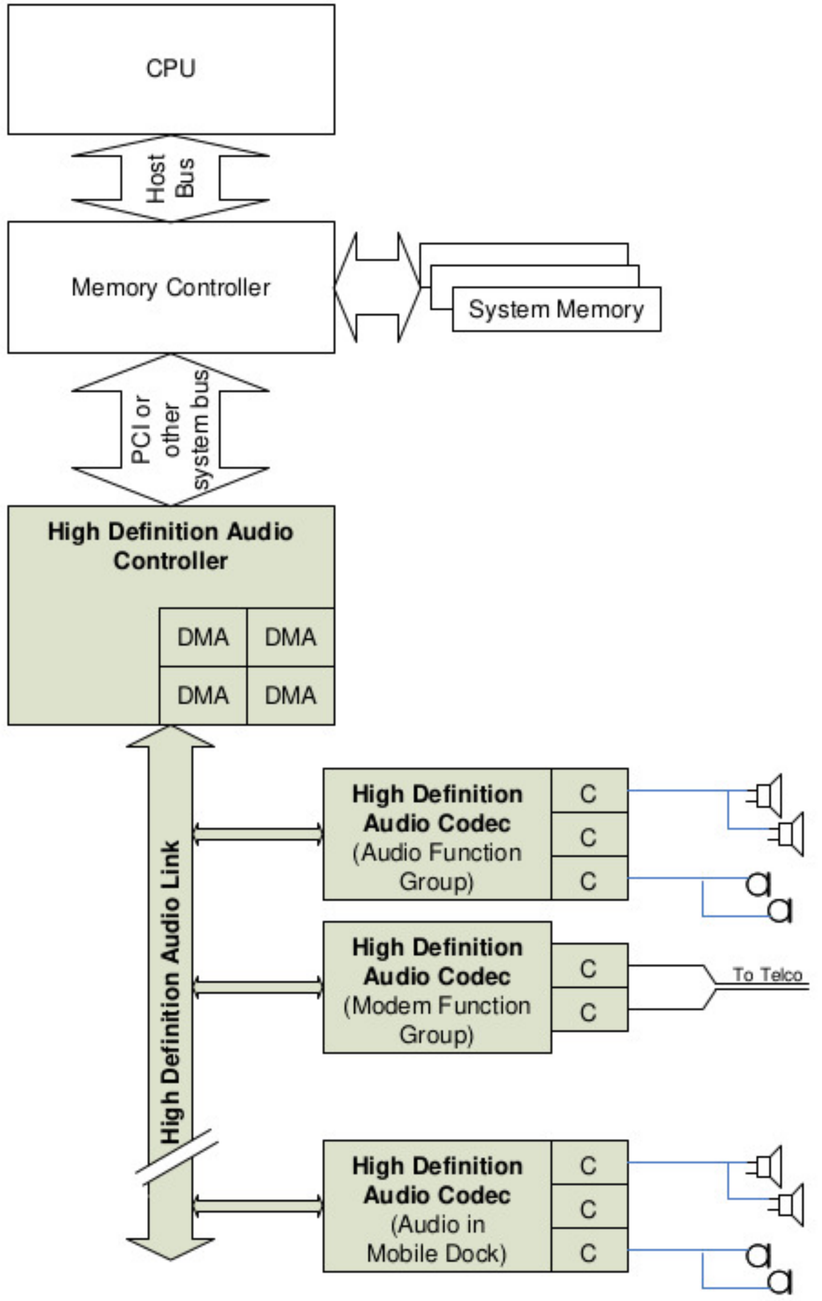
# Intel HD Audio - Architektur

## Grundlagen

- Controller (Register, DMA)
  → Schnittstelle zum Rechner
  → Gehirn

- Link (physische Verbindung zwischen Controller und Codecs)
  → bidirektionaler Nervenstrang

- Codec (DACs/ADCs)→ Schnittstelle zur Außenwelt
  → Mund und Ohren



# Intel HD Audio - Controller

## Grundlagen

- Nutzt als Bus-Master First-Party-DMA für verschiedene Zwecke (Register und diverse Buffer, insbesondere Audiobuffer)
- Anzahl der Register hängt von der Anzahl der verbauten DMA-Engines ab
- Kommunikation zwischen Treiber und Controller über Register (Initialisierung und Konfiguration)
- Kommunikation zwischen Treiber und Codec nur über Controller möglich: entweder über Immediate Command Input and Output Registers oder über Command Outbound Ring Buffer (CORB) und Response Inbound Ring Buffer (RIRB)

# Intel HD Audio - Link

## Grundlagen

- serialisierte Daten (Audiodaten und Nachrichten zwischen Treiber und Codec)
- bidirektional
- Übertragungsrate: 48 kHz → 20,83 µs zwischen zwei Frames

# Intel HD Audio - Codec

## Grundlagen

- Bezeichnung für jegliches Gerät, welches an den Link angeschlossen ist
- im Kontext der Bachelorarbeit: besitzt mindestens einen DAC → Audiocodec (kein Modem Codec oder Vendor Defined Codec) und nicht rein digital (HDMI, DisplayPort, SPDIF, …)
- Modulare Architektur (Nodes, Widgets)
- Jede Node besitzt eindeutige Adresse (Codec Address, Node ID)
- Jede Node besitzt read-only Parameters und read-write Controls
- Kommunikation zwischen Treiber und Nodes über Controller (Commands, Responses)
- Topologie eines Codecs initial unbekannt → muss von Treiber durch Befragung der einzelnen Nodes ermittelt werden

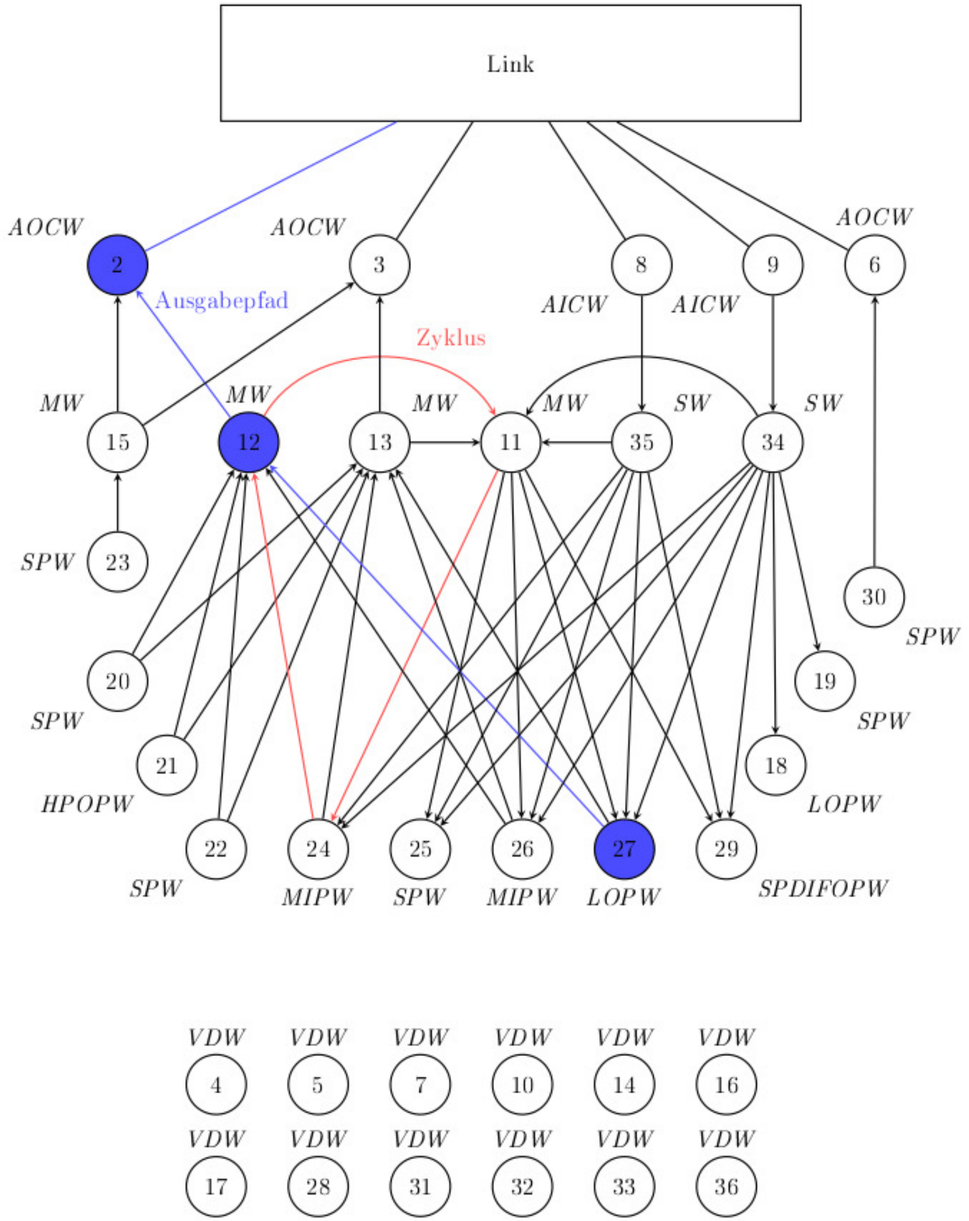
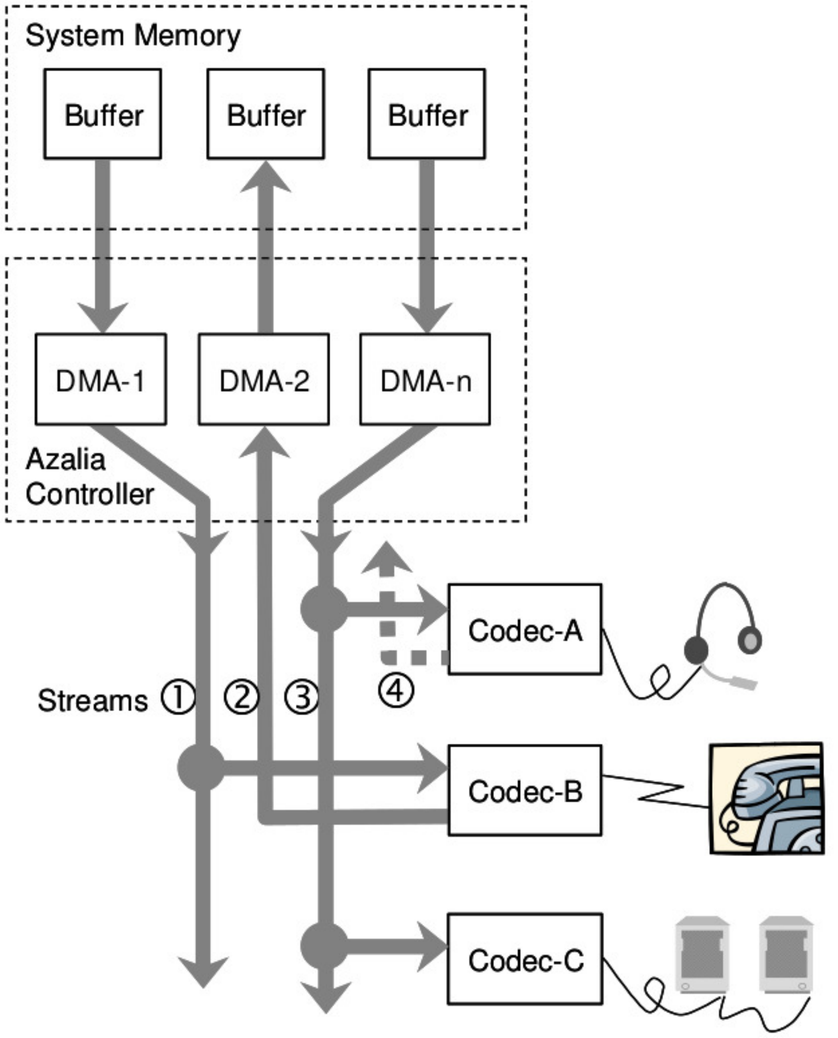



Abbildung 6.1: Verknüpfungsgraph der *Widgets* innerhalb des *Codecs* der in den Testrechner verbauten integrierten Soundkarte

# Intel HD Audio - Streams

## Grundlagen

- Audiodaten auf dem Link werden durch Streams organisiert
- Stream: Verbindung zwischen mindestens zwei Audiobuffern und einem oder mehreren Codecs
- 1 Stream ↔ 1 DMA-Engine ↔ 1 Stream ID ↔ 1 Buffer Descriptor List
- entweder Input-Stream oder Output-Stream
- Stream Format (Samplerate, Bittiefe, bis zu 16 Channels)



# Intel HD Audio - Programmiermodell

## Grundlagen

- Initialisierung des PCI Interfaces (Gerät finden, Bits im PCI Configuration Space setzen, Interrupt Line verbinden, Mapping der Register)
- Start und Konfigurierung des Controllers
- Ermittlung der Topologie des Codecs
- Erstellung eines Streams (Buffer allozieren, DMA-Engine über Register konfigurieren)
- Konfigurierung des Codecs für die Audiowiedergabe (inklusive Finden eines Ausgabepfads)
- Start des Streams

# Struktur des Treibers

## Implementierung

- Objektorientierter Ansatz (Entitäten → Enums, Structs, Funktionalität → Methoden, Funktionen)
- 4 Module: API, PCI, Controller und Codec
- Offene Schichtenarchitektur:
  API → PCI, Controller, Codec
  Controller → Codec
- Über 70 Structs und Enums (>80% davon Commands, Responses und Payloads für die Kommunikation zwischen Treiber und Codec) und fast 250 Funktionen und Methoden (>60% davon Methoden auf dem Struct Controller

# Integration in das D³OS

## Implementierung

- Steuerung der Soundkarte über Methoden des im API-Modul definierten Structs IntelHDAudioDevice → API
- Treiberinitialisierung im Rahmen des Systemstarts (Funktion start im boot.rs)
- Dort Aufruf des Konstruktors von IntelHDAudioDevice → Referenz auf erzeugte Instanz wird in statischer Variable gespeichert
- Referenz kann von jedem Prozess im Kernel-Adressraum aus durch Aufruf der Funktion intel_hd_audio_device eingeholt werden → Soundkarte kann von jedem Prozess im Kernel-Adressraum bedient werden

# API-Modul

## Implementierung

- Orientiert sich an Struktur bereits im D³OS vorhandener Treiber (APIC, PS/2)
- Interrupt Handler
- Struct IntelHDAudioDevice (einziges öffentliches Objekt im einzigen öffentlichen Modul des Treibers) → Soundkarte nur über diese API bedienbar
- IntelHDAudioDevice muss Traits Sync und Send implementieren, da es Pointer auf Adressen im Kernel-Adressraum enthält, welche in anderen Adressräumen ungültig wären

# PCI-Modul

## Implementierung

- bündelt alle PCI-Funktionalität (Gerät finden, Bits im PCI Configuration Space setzen, Interrupt Line verbinden, Mapping der Register)
- eigentlich BS-Funktionalität

# Controller-Modul

## Implementierung

- Enthält zentrales Struct Controller, über welches der Treiber die Soundkarte steuert
- Controller implementiert externalisierte Konstruktoren der im Codec-Modul definierten Structs Codec, FunctionGroup und Widget, da die Topologie eines Codecs nur mithilfe des Controllers ermittelt werden kann
- Enthält Structs zur Erzeugung eines Streams (Stream, AudioBuffer, BufferDescriptorList, StreamFormat, …)

# Codec-Modul

## Implementierung

- Enthält Structs zur Repräsentation der Topologie eines Codecs (Codec, FunctionGroup, Widget, NodeAddress)
- Enthält Implementationen der für die Kommunikation zwischen Treiber und Codec benötigten Commands und Responses

# Herausforderungen bei der Treiberentwicklung

## Evaluation

- Komplexität und Offenheit der Intel HD Audio Architektur (wenige Sekundärquellen, modulare Codec-Architektur → kein universeller Algorithmus zur Konfigurierung eines Codecs → Treiber unterstützt momentan genau ein Modell)
- Testen auf virtualisierter Hardware in QEMU (Registerzugriffe funktionierten, aber Controller der virtualisierten Soundkarte interagierte nicht mit anderen DMA-Bereichen → Testen auf echter Hardware notwendig)
- Das noch junge D³OS (Betriebssystemfunktionalität für Konfigurierung des PCI-Interfaces und No-Cache-Allocation musste selbst geschrieben werden, Komplexität des D³OS konnte nur bedingt ausgeblendet werden)

# Funktionalität des Treibers

## Evaluation

- Benchmarks nicht sinnvoll, da Audio mit konstanter Datenrate übertragen wird
- Funktionierende Audiowiedergabe beweist implizit korrekte Umsetzung des Programmiermodells
- Für nicht für die Audiowiedergabe benötigte Buffer (CORB und RIRB, DMA Postition Buffer) wurden Testfunktionen geschrieben
- Aufruf der Demo-Funktion aus dem boot.rs heraus (also von außerhalb der Module des Treibers) zeigt, dass API funktioniert

# Demo

- Output-Stream
- Samplerate: 48 kHz, Bittiefe: 16 Bit, Channel: 2 (stereo)
- 8 Audiobuffer mit je 64 * 4096 Byte
  → Abspieldauer pro Buffer: 64 * 4096 Byte / 48 kHz / 16 Bit / 2 ≈ 1.36533s
- In jedem Audiobuffer Sägezahnschwingungen in PCM-Darstellung, Frequenz verdoppelt sich jedes Mal: 50 Hz, 100 Hz, 200 Hz, 400 Hz, 800 Hz, 1600 Hz, 3200 Hz, 6400 Hz

# Zusammenfassung

## Fazit und Ausblick

- Der entwickelte Treiber stellt dem D³OS eine API zur Verfügung, über welche die Soundkarte aus beliebigen Prozessen im Kernel-Adressraum heraus bedient werden kann
- Audiowiedergabe funktioniert → Ziel erreicht
- Für die Treiberentwicklung benötigte Kenntnisse:
  Programmierkenntnisse in Rust
  allgemeine Kenntnisse der Betriebssystemprogrammierung (DMA, virt. SV, …)
  spezifische Kenntnisse über das D³OS
  Kenntnisse über den PCI-Bus und das PCI-Interface
  Grundkenntnisse in Signalverarbeitung
  sehr detaillierte Kenntisse über die Intel HD Audio Spezifikation

# Mögliche Erweiterungen
## Fazit und Ausblick

- Interrupts
- Input-Streams
- Implementation von Controller sowie Commands und Responses vervollständigen
- Alle Widgets unterstützen
- CORB und RIRB statt Immediate Command Input and Output Registers nutzen
- Zugriffe auf CORB und RIRB sowie Audiobuffer erleichtern (Dateiformat für PCM-Daten, welches alle Sampleraten und Bittiefen unterstützt)
- Synchronisation (von Streams und Controllern), Striping, Power Management
- mehr Modelle unterstützen